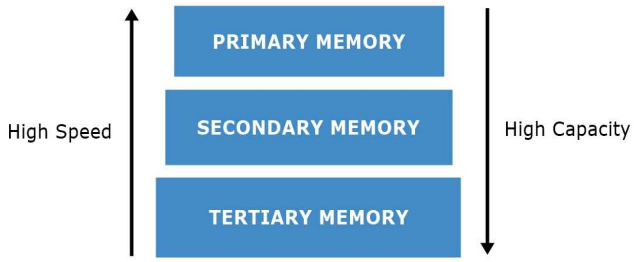
Čuvanje i transport podataka

Baze podataka čuvaju se u formatu fajlova koji u sebi sadrže neke zapise. Na fizičkom nivou, podaci se čuvaju u elektromagnetnom formatu na nekom uređaju za skladištenje, odnosno u nekoj memoriji. Memorija se uglavnom razlikuje po kapacitetu i brzini pristupa. Na osnovu ovog kriterijuma, memoriju uopšteno možemo podeliti na tri vrste, kao što je prikazano na slici 3.1).



Slika 3.1. Opšta podela uređaja za skladištenje

- **Primarna memorija** nalazi se na matičnoj ploči i direktno je dostupna procesoru. Tu spadaju interna memorija (registri), brza (keš) memorija i glavna memorija (RAM – Random Access Memory). Primarna memorija je obično vrlo mala i izvršavanje u njoj je izuzetno brzo. Za primarnu memoriju je neophodno stalno napajanje kako bi se održalo njeno stanje. U slučaju nestanka struje, svi podaci se gube.
- **Sekundarna memorija** skladišti podatke za buduće korišćenje ili kreira rezervne kopije podataka. Ova memorija ne uključuje uređaje koji su deo procesora ili matičnog seta. U sekundarnu memoriju spadaju magnetni diskovi, optički diskovi (CD, DVD itd.), hard diskovi, fleš diskovi, magnetne trake i sl.
- **Tercijarna memorija** koristi se za skladištenje ogromne količine podataka. Prema računarskom sistemu, ova memorija predstavlja spoljnu memoriju, zbog čega je rad na podacima u njoj sporiji. Ova memorija uglavnom koristi optičke diskove i magnetne trake.

Računarski sistem ima dobro definisanu hijerarhiju memorije. Procesor ima direktan pristup glavnoj memoriji, kao i ugrađenim registrima. Međutim, vreme pristupa glavne memorije je mnogo sporije od procesora. Problem ove neusaglašenosti rešava se uvođenjem cache (keš) memorije. Ona pruža najbrže vreme pristupa i sadrži podatke kojima procesor najčešće pristupa. Memorija sa najbržim pristupom je najskuplja. Veći uređaji za skladištenje nude malu brzinu i jeftiniji su, ali mogu da skladište ogromne količine podataka.

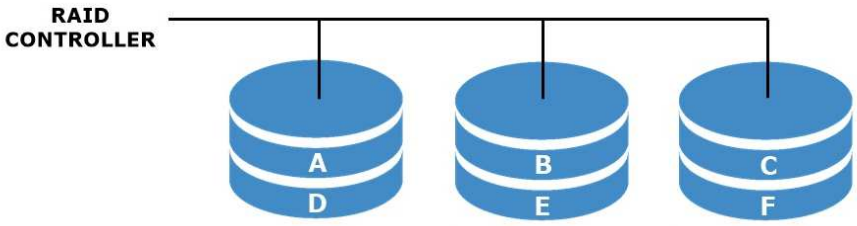
RAID

Hard disk (magnetni disk) je najšire upotrebljavani uređaj za sekundarno skladištenje podataka – odnosno, najzastupljenija sekundarna memorija. Sačinjen je od tvrdih diskova koji su obloženi magnetizacijskim materijalom i postavljeni vertikalno na vreteno. Glava za čitanje, odnosno upisivanje podataka pomera se između diskova i vrši magnetizaciju i demagnetizaciju podataka na osnovu binarnih zapisa 1 i 0 na diskovima.

Ploča hard diska u sebi ima mnogo koncentrisanih krugova koji su podeljeni u sektore od kojih svaki obično sadrži 512 bajtova. Hard diskovi se formatiraju po dobro definisanom redosledu za efikasno čuvanje podataka. U ove svrhe koristi se RAID tehnologija.

RAID (redudant array of independent discs) predstavlja tehnologiju za povezivanje više sekundarnih uređaja za skladištenje podataka u jedinstveni medijum. Ova tehnologija omogućava povezivanje nekoliko diskova u jedinstveni niz radi postizanja različitih ciljeva. U zavisnosti od cilja koji želimo da postignemo, razlikujemo nekoliko RAID nivoa.

RAID 0 – na ovom nivou je implementiran raščlanjeni niz diskova. Ovde se podaci dele na blokove i blokovi se raspoređuju na diskove. Svaki disk prima blok podataka za paralelno čitanje ili upisivanje. Na taj način se poboljšavaju brzina i performanse uređaja za skladištenje podataka, odnosno memorije. Na ovom nivou nema pariteta i rezervnih kopija.

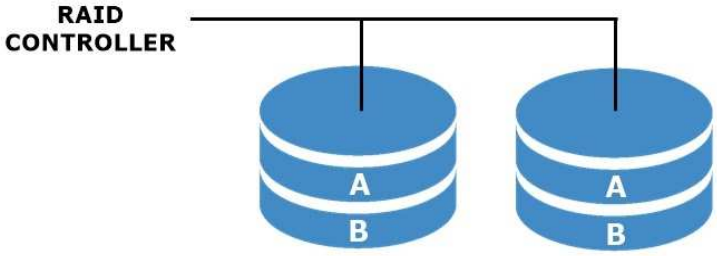


Slika 3.2. Upisivanje podataka u sekundarnu memoriju korišćenjem RAID 0 tehnologije

Napomena

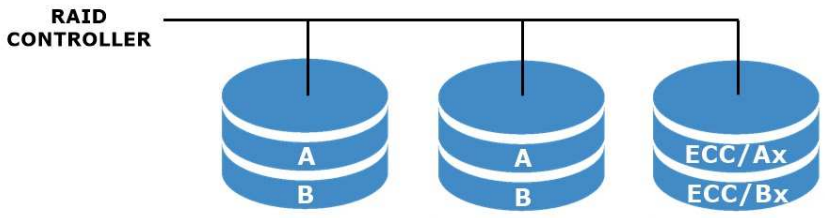
Paritet, odnosno bit pariteta, u programiranju predstavlja bit za proveru prosleđenog binarnog stringa za upis podataka. On se dodaje na početku binarnog stringa i stara se za to da broj bitova vrednosti 1 postane paran ili neparan. U zavisnosti od onoga što želimo da postignemo, razlikujemo parni paritet i neparni paritet.

RAID 1 koristi tehniku ogledala. Kada se podaci pošalju RAID kontroleru, on šalje kopiju podataka na sve ostale diskove u nizu. Ovaj nivo omogućava stopostotnu redundantnost podataka u slučaju kvara.



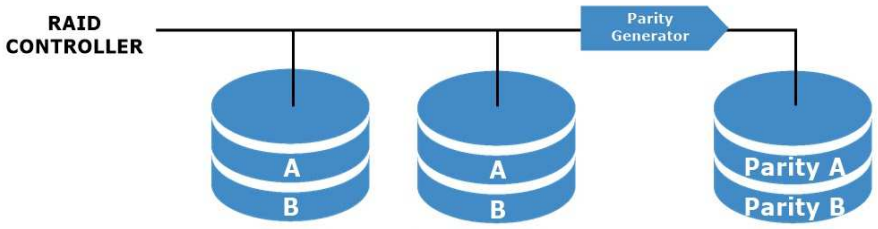
Slika 3.3. Upisivanje podataka u sekundarnu memoriju korišćenjem RAID 1 tehnologije

RAID 2 koristi kod za ispravljanje grešaka (*ECC – error correction code*) upotrebom *Hamming* udaljenosti za podatke raščlanjene na različitim diskovima. I ovde se, kao i na nultom nivou, svaki bit podatka čuva na različitom disku i *ECC* kodovi se čuvaju na različitom setu diskova. Zbog svoje složene strukture i visokih troškova, ovaj nivo nije javno dostupan.



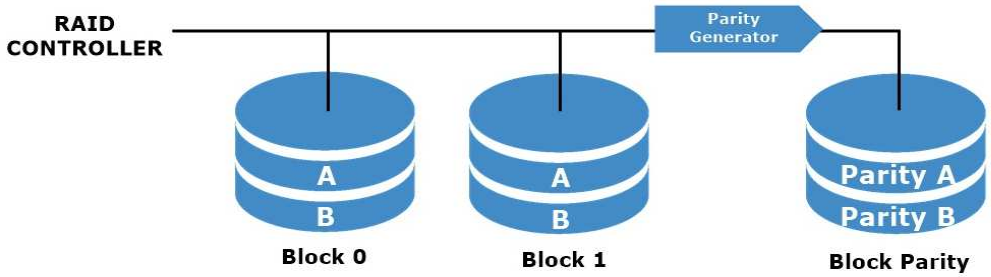
Slika 3.4. Upisivanje podataka u sekundarnu memoriju korišćenjem RAID 2 tehnologije

RAID 3 preusmerava podatke na više diskova. Bit pariteta koji je generisan za svaku reč čuva se na odvojenom disku. Ovom tehnikom se prevazilaze greške na jednom disku.



Slika 3.5. Upisivanje podataka u sekundarnu memoriju korišćenjem RAID 3 tehnologije

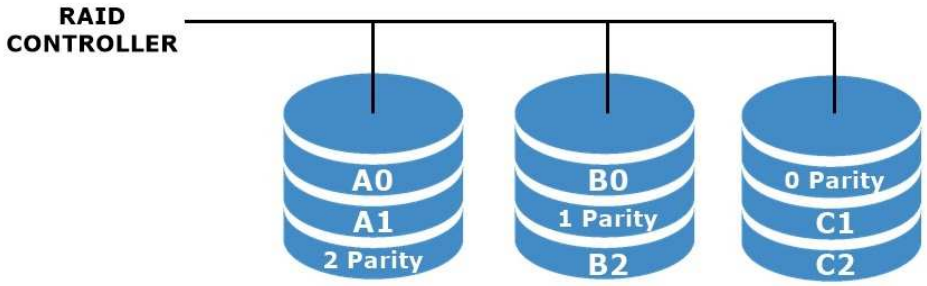
RAID 4 čitav blok podataka zapisuje na diskove podataka, a zatim se paritet generiše i čuva na drugom disku. Za razliku od trećeg nivoa, koji podatke raščlanjuje po bitovima, ovaj nivo podatke rašlanjuje po blokovima.



Slika 3.6. Upisivanje podataka u sekundarnu memoriju korišćenjem RAID 4 tehnologije

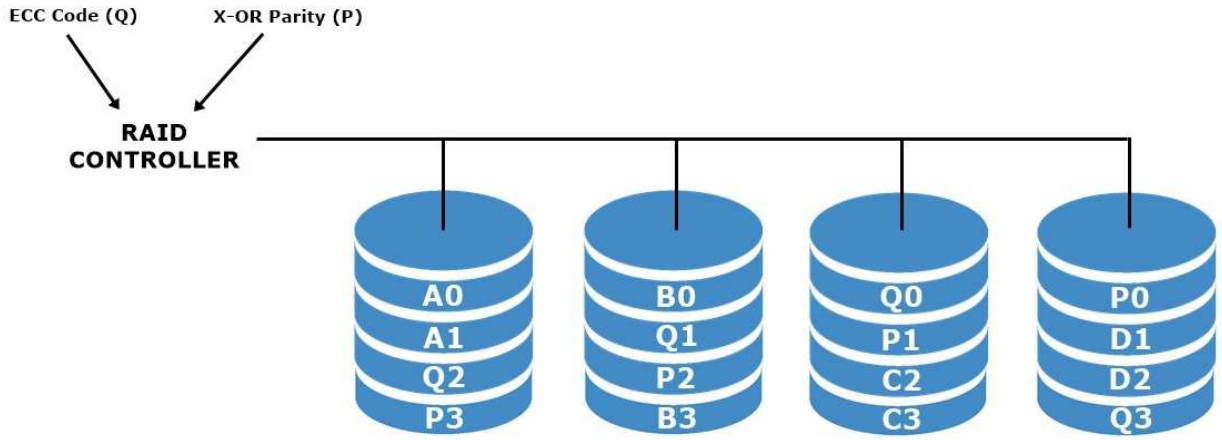
RAID 3 i RAID 4 zahtevaju minimum tri diska da bi njihova implementacija bila moguća.

RAID 5 piše čitave blokove podataka na različite diskove, ali su bitovi pariteta generisani za raščlanjene blokove podataka distribuirani na sve diskove umesto da budu čuvani na jednom namenskom disku.



Slika 3.7. Upisivanje podataka u sekundarnu memoriju korišćenjem RAID 5 tehnologije

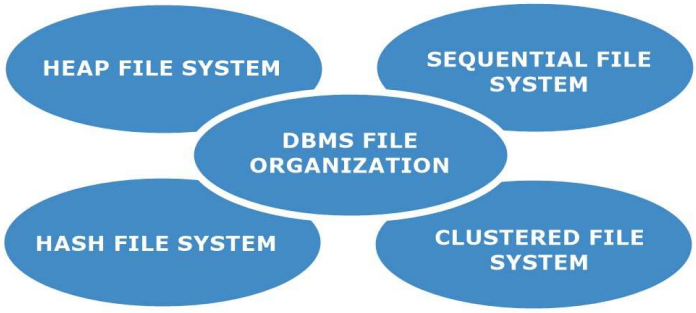
RAID 6 predstavlja poboljšanje petog nivoa. Na ovom nivou se generišu i distribuiraju dva nezavisna pariteta na raspodeljeni način među više diskova. Na ovaj način se obezbeđuje dvostruka intolerancija na greške. Za implementaciju su neophodna minimum četiri diska.



Slika 3.8. Upisivanje podataka u sekundarnu memoriju korišćenjem RAID 6 tehnologije

Organizacija fajlova i operacije nad njima

Relativni podaci i informacije se zajedno čuvaju u formatu fajlova. U ovom slučaju fajl predstavlja niz zapisa koji se skladište u binarnom formatu. Disk za skladištenje podataka formatiran je u nekoliko blokova na koje se mogu skladištiti zapisi. Zapisi koji su na fajlu mapirani su na te diskove. Kako će podaci biti mapirani – zavisi od organizacije fajlova.



Slika 3.9. Načini organizovanja fajlova u sekundarnoj memoriji

Organizacijom fajlova mapiranih na disku bavi se sistem za upravljanje bazama podataka. Razlikujemo četiri načina organizovanja fajlova:

- **Heap organizacija fajlova** podrazumeva da operativni sistem dodeljuje memorijsku lokaciju tom fajlu bez ikakvih dodatnih preračunavanja. Zapisi tog fajla mogu se skladištiti bilo gde na dodeljenoj memorijskoj lokaciji. Odgovornost za upravljanje fajlovima pripada softveru. Fajl skladišten na heapu ne dozvoljava ručno dodeljivanje lokacije, sekvenciranje i indeksiranje.
- **Sekvencijalna organizacija fajlova** podrazumeva da svaki zapis datoteke sadrži polje podatka (atribut) koje predstavlja njegov jedinstveni identifikator. U ovoj organizaciji, u fajlu se zapisi postavljaju određenim redosledom, na osnovu jedinstvenog primarnog ključa ili ključa za pretragu. U praksi, fizički je nemoguće skladištiti sve podatke u redosledu.
- **Hash organizacija fajlova** koristi izračunavanje funkcije hash na nekim poljima zapisa. Autput hash funkcije određuje lokaciju bloka diska gde će zapisi biti smešteni.
- **Klasterirana organizacija fajlova** – u ovom mehanizmu srodni zapisi iz jedne ili više relacija čuvaju se na istom diskovnom bloku, odnosno, redosled zapisa se ne zasniva na primarnom ključu ili ključu za pretragu.

Operacije nad fajlovima baza podataka mogu se uopšteno podeliti na dve kategorije:

- **operacije ažuriranja** menjaju vrednost podataka dodavanjem, brisanjem ili izmenom;
- **operacije prikaza** prikazuju željene podatke na osnovu zadatog kriterijuma.

Osim kreiranja i brisanja fajlova, postoji još nekoliko osnovnih operacija koje se mogu izvršiti nad njima.

- **Open** – fajl se moze otvoriti u režimu čitanja ili u režimu pisanja. U režimu čitanja, sistem ne dozvoljava nikome da izvrši izmenu nad podacima. Drugim rečima, podaci se u ovom režimu mogu samo čitati i deliti među drugim entitetima. U režimu pisanja, podaci se mogu čitati, upisivati i menjati, ali ne i podeliti.
- **Locate** – svaki fajl ima pokazivač koji daje informaciju o tome gde operacija čitanja ili pisanja treba da bude izvršena. Pokazivač se u skladu sa tim može prilagoditi, odnosno pomeriti napred ili nazad.

- **Read** – pozivanjem ove funkcije otvaramo fajl u režimu čitanja, dok pokazivač fajla upućuje na njegov početak. U okviru nje postoje i opcije kojima korisnik može reći sistemu gde želi da postavi pokazivač u trenutku otvaranja fajla.
- **Write** – ova funkcija omogućava uređivanje sadržaja fajlova. To mogu biti brisanje, dodavanje ili modifikacija. Pokazivač se može locirati u trenutku otvaranja ili zadati ukoliko sistem to dozvoljava.
- **Close** – najvažnija operacija sa stanovišta operativnog sistema. Kada se generiše zahtev za zatvaranje fajla, operativni sistem čuva podatke, oslobađa bafer i uklanja sve lokote, o kojima će biti više reči u nastavku.

Indeksiranje

Znamo da svaki podatak koji se upisuje u fajl predstavlja jedan zapis. Kako bi svaki od zapisa bio jedinstven, neophodno je da sadrži polje sa jedinstvenom vrednošću koja ga određuje. To se postiže indeksiranjem.

Indeksiranje je tehnika struktura podataka za efikasno pronalaženje zapisa na osnovu nekih atributa nad kojima je vršena. Ova tehnika se može poistovetiti sa obeležavanjem stranica u knjigama.

Postoji više vrsta indeksiranja.

- **Primarni indeks** definisan je u fajlu sa uređenim podacima. Uređivanje fajla vrši se određivanjem polja koje će biti ključ. Ključ predstavlja primarnu vezu sa ostalim podacima.
- **Sekundarni indeks** može se generisati iz polja koje je kandidat za ključ i ima jedinstvenu vrednost u svakom zapisu.
- **Indeks klasteriranja** definiše se u uređenom fajlu podataka. Podaci su određeni poljima koja nisu ključ.

Uređeno indeksiranje možemo podeliti u dve kategorije.

Gusto indeksiranje daje jedinstveni zapis indeksa za svaku vrednost ključa za pretragu u bazi podataka. Ovo ubrzava pretragu, ali zahteva više memorije za skladištenje svih zapisa indeksa. Zapisi indeksa sadrže vrednost ključa za pretraživanje i pokazivač na stvarni zapis na disku. Radi lakšeg razumevanja, na sledećoj slici 3.10. prikazano je kako gusto indeksiranje izgleda na primeru tabele prve španske fudbalske lige (slika 3.10).

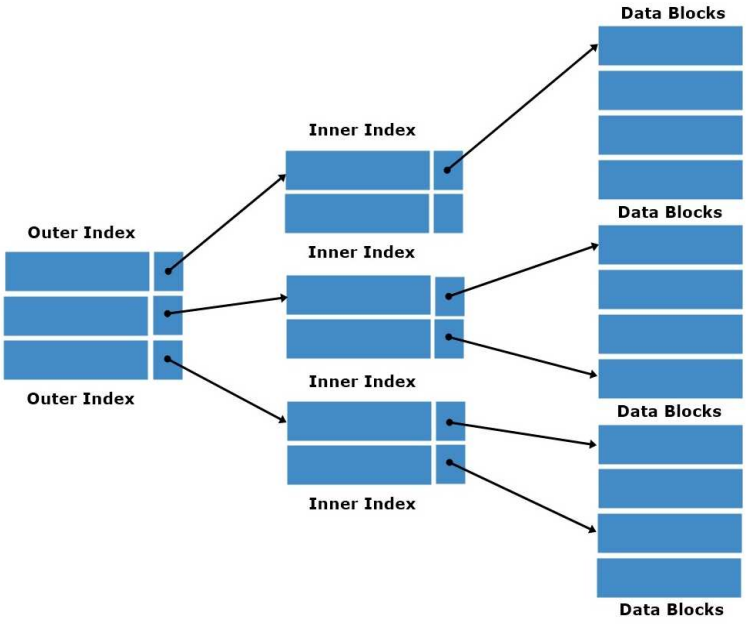
Slika 3.10. Primer gustog indeksiranja

Retko indeksiranje – indeksni zapisi se ne stvaraju za svaki ključ za pretragu. Oni takođe sadrže ključ za pretraživanje i pokazivač na podatke na disku. Da bismo pretražili zapis, koristimo indekse kako bismo došli do lokacije na kojoj je podatak skladišten. Ako se podatak ne nalazi na toj lokaciji, sistem nastavlja sekvencijalnu pretragu dok ne nađe željeni podatak. U nastavku prikazujemo kako na našem primeru izgleda retko indeksiranje (slika 3.11).

Slika 3.11. Primer retkog indeksiranja

Višeslojni indeksi čuvaju se na disku zajedno sa fajlovima baza podataka. Veličina indeksa raste zajedno sa veličinom baze podataka. Zbog potrebe za brzim pristupom podacima, indeksi baza podataka se skladište u glavnoj memoriji.

Kada je veličina indeksa veća od one koja može da se skladišti na memoriji, dolazi do višestrukog pristupa disku i raščlanjenja indeksa na više slojeva. Raščlanjivanje se vrši do najmanje veličine indeksa koji je moguće skladištiti na jednom diskovnom bloku koji se lako može smestiti bilo gde u glavnoj memoriji.

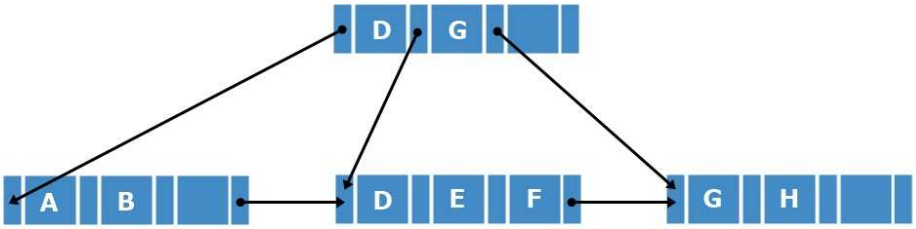


Slika 3.12. Prikaz strukture višeslojnih indeksa

B^+ stablo je uravnoteženo stablo binarnog pretraživanja koje prati višeslojni format indeksa. Koncept stabala zasniva se na čvorovima koji predstavljaju osnovnu jedinicu skladištenja podataka u računarstvu. Početni čvor u stablu naziva se koren. Svaki čvor u stablu pretraživanja koji nema nijedan čvor ipod sebe (dete) naziva se list. Listovi u sebi ne sadrže podatke. Osim toga, čvorovi su povezani pomoću spregnute liste, što daje mogućnost kako slučajnog tako i sekvencijalnog pristupa.

Postoji nekoliko pravila strukture kojih se ovo stablo pridržava:

- svi listovi stabla se nalaze na istoj dubini u odnosu na koren;
- koren stabla, ako nije list, ima minimum dva pokazivača, od kojih jedan pokazuje na levi, a drugi na desni čvor (dete);
- svaki čvor u stablu koji nije koren ima najmanje *n+1* dece;
- svaki list u stablu sadrži pokazivač na susedni list i formira spregnutu listu.



Slika 3.13. Prikaz strukture B^+ stabla pretraživanja

Upisivanje podataka u stablo vrši se odozdo nagore. Ako čvor ne može da prihvati sve podatke, uvodi se novi čvor i vrši se particionisanje podataka na *i* i *i+1* lokaciji i ključ se prosleđuje čvoru koji je roditelj.

Brisanje podataka iz stabla se vrši odozdo nagore. Ako obrišemo čvor koji nije list, njega će zameniti sledbenik sa leve strane. Ako čvor na levoj strani ne postoji, uzima se desni.

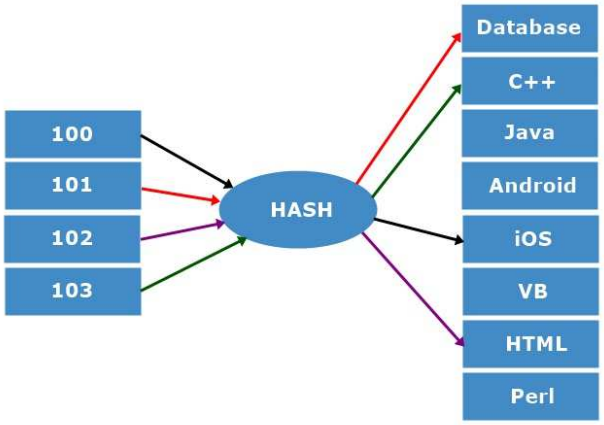
Hashovanje

Kod velikih baza podataka gotovo je nemoguće pretražiti sve indeksne vrednosti po njihovim nivoima u cilju prikazivanja željenih podataka. **Hashovanje** predstavlja efikasnu tehniku za računanje direktne lokacije traženog podatka na disku bez korišćenja tehnike indeksiranja. Ova tehnika koristi hash funkcije koje, pomoću ključa pretraživanja kao parametra, generišu adresu zapisa podatka.

Prema načinu organizacije, razlikujemo dva načina hashovanja:

- **paket**, gde hash fajl skladišti podatke u formatu paketa; svaki paket predstavlja jedinicu skladištenja i obično skladišti jedan blok diska na kojem mogu biti smešteni jedan ili više zapisa;
- **hash funkcija** – funkcija **h** mapira ceo set ključeva pretrage **K** na adrese gde se nalaze konkretni podaci.

Kada prosledimo ključ pretrage kao parametar, funkcija za njega uvek računa istu adresu. Broj paketa uvek ostaje nepromenjen.



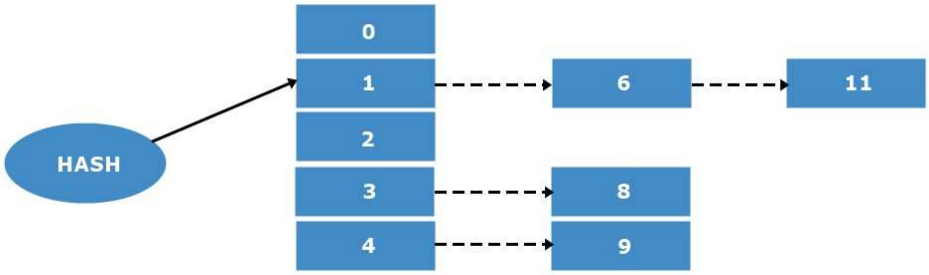
Slika 3.14. Primer hashovanja za četiri memorijske lokacije

Hashovanje razlikuje nekoliko osnovnih operacija:

- **upisivanje** – statički unos zapisa pomoću funkcije **x** koja pomoću parametra ključ pretrage izračunava adresu gde će se zapis čuvati;
- **pretraga** – može se koristiti za pronalaženje željenog zapisa pomoću adrese ili pronalaženje same adrese;
- **brisanje** – jednostavno pretraživanje koje rezultuje brisanjem podatka.

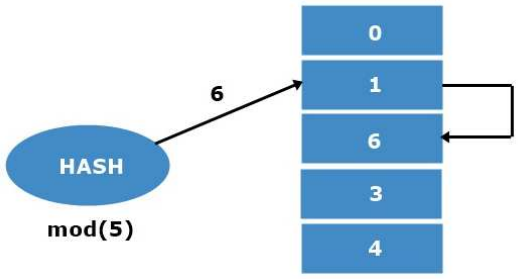
Kao glavni problem statičkog hashovanja javlja se takozvani ***bucket overflow***, odnosno prelivanje podataka, do kojeg dolazi kada u ciljanom paketu nema dovoljno memorije. Postoje dva rešenja ovog problema:

- **lanac prelivanja (chain overflow)** – kada se paket napuni, alocira se novi za isti hash rezultat i povezuje se sa prethodnim paketom; ovaj mehanizam se naziva još i **zatvoreno heširanje**;



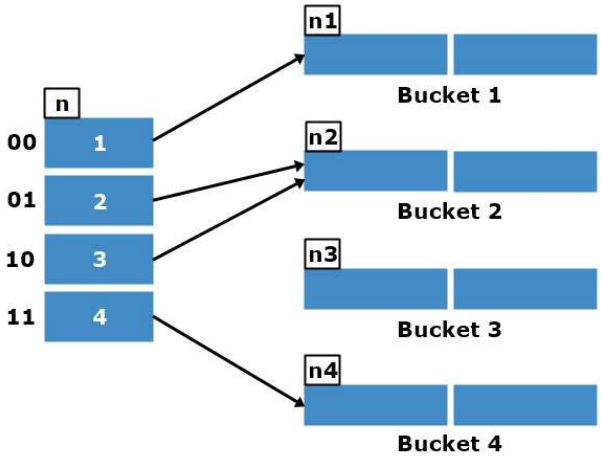
Slika 3.15. Prikaz zatvorenog heširanja – chain overflow

- **linearno sondiranje** – kada hash funkcija generiše adresu na kojoj su podaci već skladišteni, njoj se dodeljuje sledeći slobodni paket; ovaj mehanizam se naziva još i **otvoreno heširanje**; na slici 3.16. prikazano je kako otvoreno heširanje izgleda na primeru dodele pet memorijskih lokacija.



Slika 3.16. Prikaz otvorenog heširanja (linearno sondiranje)

Problem kod statičkog heširanja je taj što se ne povećava, odnosno ne smanjuje, zajedno sa povećanjem ili smanjenjem baze podataka. **Dinamičko heširanje** pruža mehanizam u kojem se paketi podataka dodaju i uklanjaju dinamički i na zahtev. Ovo heširanje se naziva još i **napredno heširanje**. U dinamičkom heširanju, hash funkcija je napravljena tako da računa veliki broj vrednosti, od kojih su samo neke korišćene inicijalno.



Slika 3.17. Prikaz dinamičkog (naprednog) heširanja

Prefiks celokupne hash vrednosti se uzima kao hash indeks, dok se jedan mali deo uzima za računanje adresa paketa. Svaki hash indeks ima vrednost dubine koja označava koliko bitova se koristi za računanje hash funkcije. Ovi bitovi mogu adresirati 2n paketa. Kada se potroše svi ovi bitovi, odnosno kada su paketi puni, vrednost dubine se linearno povećava i dva puta se dodeljuju paketi.

Heširanje podržava nekoliko osnovnih operacija:

- **upiti** – korišćenjem bitova koji predstavljaju vrednosti dubine hash indeksa računaju adrese paketa;
- **ažuriranje** – omogućava izmene na podacima korišćenjem upita;
- **brisanje** – pronalazi podatke iz upita i briše ih;
- **upisivanje** – računa adrese paketa i upisuje ih po sledećem algoritmu:
    - ako je paket već pun:
        - dodaje još paketa;
        - dodaje još bitova na hash funkciju;
        - ponovo izračunava adresu;
    - u suprotnom:
        - dodaje podatak u paket;
        - ako su svi paketi puni, potrebno je izvršiti uklanjanje statističkog heširanja.

Hešing nije povoljan kada radimo sa podacima koji su organizovani u nekom redosledu i upiti zahtevaju niz podataka. Hash se najbolje ponaša kada su podaci diskretni i slučajni. Za razliku od indeksiranja, hešing ima složeniji algoritam i sve operacije izvršava u stalnom vremenu.

Transakcije

Transakcija se može definisati kao grupa zadataka. Pojedinačni zadatak je minimalna procesna jedinica koja se ne može dalje deliti. Radi boljeg objašnjenja pojma transakcije, prikazaćemo kako to izgleda na primeru uplate novca sa jednog računa zaposlenog A na račun zaposlenog B. Pretpostavimo da u programu već postoji definisana klasa *Account* sa deklarisanim poljem *balance* koje predstavlja trenutno stanje na računu korisnika i funkcijama `open_account()` i `close_account()`. Za primer ćemo uzeti uplatu iznosa u vrednosti 100 dolara.

Primer računa zaposlenog A:

```
open_account(A)
old_balance = A.balance
new_balance = old_balance - 100
A.balance = new_balance
```

Primer računa zaposlenog B:

```
open_account(B)
old_balance = B.balance
new_balance = old_balance + 100
B.balance = new_balance
close_account(B)
```

Transakcija je vrlo mala jedinica programa, ali može sadržati nekoliko zadataka nižeg nivoa. Kako bi održale tačnost, potpunost i integritet podataka, transakcije prate ACID osobine:

- **Atomičnost (atomicity):** Transakcija mora biti tretirana kao „atomska" vrednost, što znači – ili će se izvršiti sve njene operacije, ili neće nijedna. U bazi podataka ne sme da postoji transakcija koja je u stanju delimične izvršenosti. Stanja transakcije moraju biti definisana pre izvršavanja ili nakon izvršavanja/pobačaja/neuspeha.
- **Konzistentnost (consistency):** Baza podataka mora ostati u doslednom stanju posle bilo koje transakcije. Nijedna transakcija ne bi trebalo da ima štetno dejstvo na podatke koji se u bazi nalaze. Ako je baza podataka bila u doslednom stanju pre izvršenja transakcije, takva mora da ostane i po njenom izvršenju.
- **Izolovanost (isolation):** ako se u sistemu izvršava više transakcija istovremeno, efekti svake pojedinačne transakcije će biti izolovani. Nijedna transakcija neće uticati na postojanje bilo koje druge transakcije.
- **Trajnost (durability):** baza podataka mora da bude dovoljno izdržljiva za sve najnovije ispravke, čak i ako sistem padne ili se ponovo pokrene. Ako transakcija izvrši promene na delu podataka, te promene u bazi ostaju trajne. Ako sistem padne nakon izvršenja transakcija, ali pre upisivanja podatka na disk, podaci će se upisati nakon ponovnog pokretanja sistema.

Jedan od bitnih koncepata kad je reč o transakcijama je i **serijabilnost**. Naime, kada operativni sistem izvršava više transakcija u okruženju sa više programa, postoji mogućnost da se instrukcije jedne transakcije prepliću sa instrukcijama neke druge, što dovodi do konflikta transakcija. Postoje dva rešenja za ovaj problem:

- **raspored** – hronološko, odnosno sekvencijalno izvršavanje transakcija; raspored može da sadrži veliki broj transakcija od kojih svaka ima svoj red izvršavanja i svoje instrukcije;
- **serijski raspored** – transakcije se poravnaju tako da se jedna izvršava prva; kada prva transakcija izvrši svoj ciklus, na redu je prva sledeća; transakcije nastavljaju izvršavanje serijski, po poravnatom redosledu.

U okruženju sa više transakcija, serijsko izvršavanje se izvršava pomoću referentnih vrednosti. Sekvencijalno izvršavanje instrukcija u transakcijama nije moguće izmeniti, ali dve transakcije mogu da izvršavaju instrukcije nasumično. Ovo izvršavanje nije konfliktno ukoliko te dve transakcije rade na različitim segmentima podataka. U slučaju da se izvršenje obavlja na istom segmentu, rezultati mogu varirati. Ove varijacije mogu dovesti do nekonzistentnosti.

Da bismo rešili ovaj problem, dozvoljavamo paralelno izvršavanje rasporeda transakcija ako su transakcije u tom rasporedu serijski podesive ili imaju neki ekvivalentan odnos.

Ekvivalentni rasporedi uključuju nekoliko vrsta:

- **ekvivalentni rezultati** – smatra se da su dve transakcije ekvivalente ako daju isti rezultat nakon izvršenja; one mogu dati isti rezultat za neku vrednost i različite rezultate za drugi tip vrednosti, zbog čega se ova ekvivalencija uglavnom ne smatra značajnom;
- **ekvivalentni pogledi** – raspored dve transakcije se smatra ekvivalentnim ako transakcije u oba rasporeda izvršavaju slične aktivnosti na sličan način;
- **ekvivalentni konflikti** – dva redosleda su konfliktna ako ispunjavaju sledeće uslove:

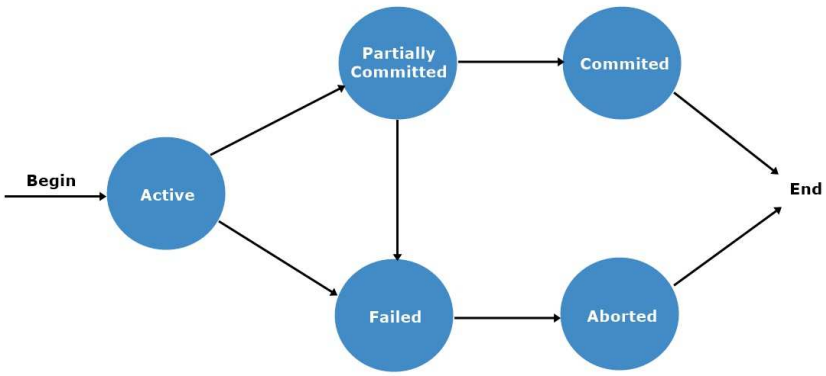
- oba pripadaju odvojenim transakcijama;
- oba pristupaju istom podatku;
- barem jedna od transakcija izvršava operaciju pisanja.

Dva rasporeda koja imaju više transakcija sa konfliktima operacija su ekvivalentno konfliktna ako i samo ako:

- oba rasporeda sadrže isti skup transakcija;
- se redosled izvršavanja konfliktnih parova održava u oba rasporeda.

Razlikujemo nekoliko **stanja transakcija**:

- **aktivna** – u ovom stanju se transakcija izvršava; ovo je početno stanje svake transakcije;
- **delimično izvršena** – transakcija dolazi u ovo stanje kada izvrši svoju poslednju operaciju;
- **neuspešna** – za transakciju kažemo da je neuspešna ako bilo koja provera izvršena od strane sistema za oporavak baze podataka ne uspe; neuspešna transakcija se dalje ne može procesirati;
- **prekinuta** – ako bilo koja provera ne uspe, transakcija dolazi u neuspešno stanje; tada upravljanje oporavkom poništava sve operacije upisivanja i vraća bazu podataka u originalno stanje u kojem je bila pre pokretanja transakcije; transakcija tada prelazi u stanje prekinute, nakon čega sistem za oporavak baze bira između dve opcije: restartovanje transakcije i njeno „ubijanje”;
- **izvršena** – u ovo stanje dolazi transakcija koja je izvršila sve svoje operacije; svi njeni efekti predstavljaju trajne promene na bazi podataka.



Slika 3.18. Tok stanja transakcija

Paralelne transakcije

U okruženju sa više programa gde se istovremeno može izvršiti više transakcija, veoma je važna kontrola njihovog paralelnog izvršavanja. Kako bismo osigurali atomičnost, izolovanost i serijabilnost, koristimo određene protokole.

Protokoli za kontrolu paralelnog izvršavanja transakcija mogu se podeliti u dve osnovne kategorije:

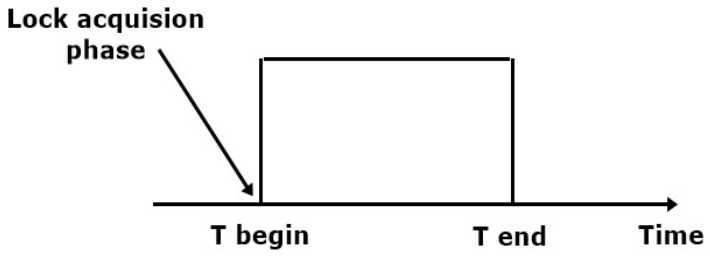
- protokoli zaključavanja;
- timestamp protokoli.

Sistemi baza podataka koji se zasnivaju na **protokolima zaključavanja** koriste mehanizam pomoću kojeg transakcija ne može vršiti operaciju čitanja ili pisanja pre nego što zaključa podatak, odnosno postavi **lokot**. Razlikujemo dve vrste protokola zaključavanja:

- **binarno zaključavanje** – podržava dva stanja, korišćenjem binarnog broja 1 za zaključano stanje i 0 za otključano;
- **ekskluzivno/deljeno** – za potrebe upisivanja postavlja se ekskluzivni lokot, jer pristup većeg broja transakcija jednom podatku dovodi do stanja inkonzistentnosti; sa druge strane, deljeno zaključavanje omogućava većem broju transakcija da postave lokot na podatak, s obzirom na to da nijedna od transakcija nema nameru da ga izmeni.

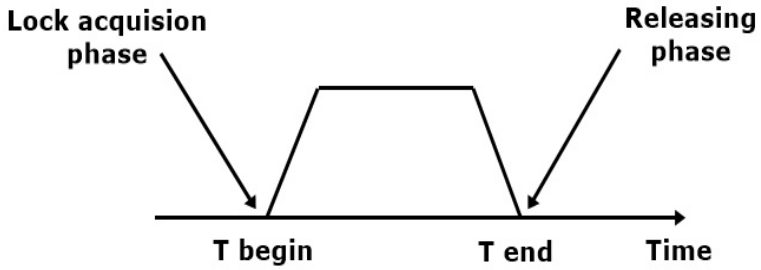
Pojednostavljeni protokoli dozvoljavaju transakcijama da postave lokot na bilo koji podatak pre izvršavanja operacije upisivanja. Nakon što je operacija upisivanja izvršena, transakcija oslobađa lokot sa podatka.

Protokoli za preispitivanje analiziraju zahteve transakcija i kreiraju listu podataka nad kojima će se vršiti zaključavanja. Pre pokretanja izvršenja, transakcija od sistema zahteva informacije o svim lokotima koji će biti postavljani na podatke kojima ona želi da pristupi. Ako su ciljani podaci slobodni, transakcija na njih postavlja lokote, izvršava upisivanje, a zatim ih oslobađa. U slučaju da neki od ciljanih podataka nije slobodan, transakcija čeka da se svi ciljani podaci oslobode da bi stupila na izvršavanje.



Slika 3.19. Vremenski prikaz izvršenja transakcije sa jednostrukim zaključavanjem

Protokoli dvostrukog zaključavanja (2PL) dele fazu izvršenja transakcije na tri dela. U prvom delu, transakcija traži dozvolu za zaključavanje koje je potrebno. Drugi deo obuhvata postavljanje svih lokota na transakcije. Čim transakcija oslobodi prvi lokot, započinje treća faza. U ovoj fazi transakcija ne može tražiti dozvolu za postavljanje novih lokota, već može samo oslobađati stare.

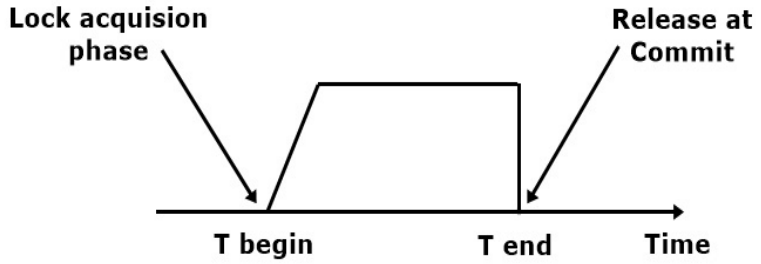


Slika 3.20. Vremenski prikaz izvršenja transakcije sa dvostrukim (2PL) zaključavanjem

Kao što možemo da primetimo na slikovnom prikazu protokola dvostrukog zaključavanja (slika 3.20), na početku transakcije grafik raste sa postavljanjem lokota na podatke. Zatim, nakon perioda izvršavanja transakcije, primećujemo opadanje na vremenskoj liniji, koje označava oslobađanje lokota sve do kraja izvršenja.

Za operaciju upisivanja podatka neophodno je da transakcija prvo zatraži pravo na čitanje, odnosno postavi deljeni lokot, a zatim unapredi to pravo u ekskluzivni lokot, odnosno pravo upisivanja.

Strogo dvostruko zaključavanje ima identičnu prvu fazu kao i 2PL protokol. Nakon sticanja prava na postavljanje lokota na podatke u prvoj fazi, transakcija nastavlja normalno da se izvršava. Jedina razlika primećuje se u oslobađanju lokota. Za razliku od običnog, strogo dvostruko zaključavanje oslobađa sve lokote istovremeno po završetku transakcije.



Slika 3.21. Vremenski prikaz izvršenja transakcije sa strogim dvostrukim zaključavanjem

Timestamp protokoli omogućavaju serijabilnost među konfliktnim transakcijama i njihovim operacijama čitanja i pisanja pomoću vremenskih oznaka. Ovaj protokol ima odgovornost da sukobljeni par zadataka transakcija izvrši u skladu sa njihovim vremenskim oznakama.

Oznake se vrše u sledećoj notaciji:

- za vremensku oznaku transakcije $\mathbf{T_i}$ koristimo oznaku $\mathbf{TS(T_i)}$;
- za vremensku oznaku čitanja nekog podatka **X** koristimo oznaku **R(X)**;
- za vremensku oznaku upisivanja nekog podatka **X** koristimo oznaku **W(X)**.

U skladu sa gore navedenom notacijom, algoritam vremenskih oznaka izgleda ovako:

- ako transakcija T_i koristi read(X) operaciju:
    - ako je TS(Ti) < W-timestamp(X)
        - operacija se odbija;
    - ako je TS(Ti) >= W-timestamp(X)
        - operacija se izvršava;
    - svi podaci se ažuriraju;
- ako transakcija T_i koristi write(X) operaciju:
    - Ako je TS(Ti) < R-timestamp(X)
        - operacija se odbija;
    - ako jeTS(Ti) < W-timestamp(X)
        - operacija se odbija i Ti se povlači;
    - u suprotnom, operacija se izvršava.

Pitanje

Indeksi baza podataka skladište se u/na:

- hard disku
- magnetnom disku
- **RAM memoriji**
- keš memoriji

Objašnjenje:

Indeksi baza podataka skladište se u glavnoj (RAM) memoriji, zbog veće brzine pristupa podacima.

Mrtvi lokot predstavlja neželjenu situaciju koja nastaje u zajedničkom resursnom sistemu sa više procesa, gde proces neograničeno čeka resurs koji je korišćen od strane drugog procesa. Ovakav konflikt nije zdrav za sistem i u ovom slučaju se transakcije ili vraćaju ili pokušavaju da se izvrše iznova.

Za primer ćemo uzeti neki skup transakcija {T_0, T_1, T_2, ...,T_n}. Pretpostavka je da T_0 želi da pristupi nekom resursu X koji koristi T_1, a T_1 želi da pristupi resursu Y koji koristi T_2. U istom trenutku, T_2 želi da pristupi resursu Z koji koristi T_0. Ovakva situacija dovodi do nedostupnosti resursa svakoj od navedenih transakcija i pojave mrtvog lokota.

Kako bi sprečio bilo kakve zastoje u sistemu, SUBP temeljno proverava sve operacije koje transakcije treba da izvrše. On pregleda operacije i analizira mogućnosti za obavljanje operacija bez konflikata. Ako utvrdi da može doći do zastoja, transakcija se nikada neće izvršiti.

Rešenje ovog problema su šeme za rešavanje zastoja koje koriste mehanizam vremenskog označavanja transakcija u cilju blagovremenog uočavanja zastoja. Razlikujemo dve šeme:

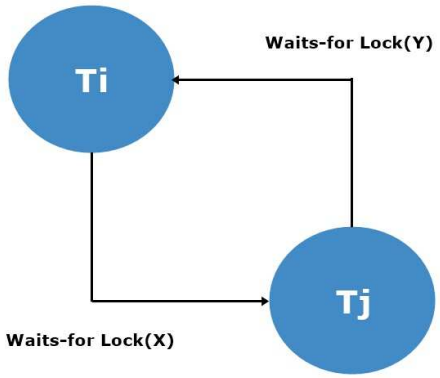
- **wait-die šema**
    - ako je TS(T_i) < TS(T_j), odnosno, ako je transakcija koja zahteva da postavi lokot mlađa od transakcije koja koristi podatak, onda ona može da čeka da se podatak oslobodi;
    - ako je TS(T_i) > TS(T_j), odnosno, ako je transakcija koja zahteva da postavi lokot starija od transakcije koja koristi podatak, onda ona „umire”, a zatim u nasumično odabranim intervalima ponovo pokušava da postavi lokot, dok njegova vremenska oznaka ostaje nepromenjena.

- **wound-wait šema**
    - ako je TS(T_i) < TS(T_j), odnosno, ako je transakcija koja zahteva da postavi lokot mlađa od transakcije koja koristi podatak, ona prisiljava transakciju koja je postavila lokot da se povuče; povučena transakcija u nasumično odabranim intervalima ponovo pokušava da postavi lokot, dok njegova vremenska oznaka ostaje nepromenjena;
    - ako je TS(T_i) > TS(T_j), odnosno, ako je transakcija koja zahteva da postavi lokot starija od transakcije koja koristi podatak, onda je ona prisiljena da čeka oslobađanje podatka.

U oba slučaja, transakcija koja uđe u kasnu fazu se prekida.

Odustajanje od transakcije nije uvek praktičan pristup. Umesto toga radije se koriste mehanizmi za izbegavanje mrtvih lokota. Jedna od najzastupljenijih i najjednostavnijih tehnika za detekciju zastoja je graf čekanja. Mana ove tehnike je mogućnost njenog korišćenja samo za male transakcije i transakcije sa malim brojem instanci.

Graf čekanja podrazumeva da se za svaku transakciju koja ulazi u sistem kreira čvor. Kada stavka T_i pokuša da postavi lokot na resurs X koji drži neka druga transakcija T_j, između ove dve transakcije se kreira usmerena putanja. Kada T_j oslobodi lokot sa podatka, putanja između ove dve transakcije nestaje i T_i postavlja svoj lokot. Sistem održava ovaj grafikon čekanja za svaku transakciju koja čeka neke stavke podataka i stalno proverava da li u grafikonu postoji neki ciklus.



Slika 3.22. Ilustracija grafa čekanja u vezi sa pojmom mrtvog lokota

Za transakcije na kojima graf čekanja nije primenljiv preporučljivo je:

- zabraniti izvršavanje transakcija nad podacima na koje je neka druga transakcija već postavila lokot;
- odabir jedne od transakcija koja će se izvršiti prva kako bi se zastoj eliminisao.

Sigurnosna kopija podataka

Nestabilne memorije poput RAM-a čuvaju sve aktivne zapise, bafere diska i podatke u vezi sa njima. Pored toga, u ovim memorijama se čuvaju i transakcije koje se trenutno izvršavaju. Kada bi se takva memorija naglo narušila, nestali bi svi zapisi i aktivne kopije baza podataka. Ovo oporavak čini gotovo nemogućim, jer se gubi sve što je neophodno za vraćanje podataka.

Za prevenciju gubitka podataka na nestabilnim memorijama koriste se sledeće tehnike:

- uvođenje kontrolnih tačaka u više faza kako bi se osiguralo povremeno čuvanje baze podataka;
- periodično čuvanje baze podataka na stabilnoj memoriji koja takođe može sadržati zapise, aktivne transakcije i bafere;
- korišćenje oznake <dump> kao markera svaki put kada se sadržaj baze podataka premesti iz nestabilne memorije u stabilnu.

Kada se sistem oporavi od kvara, može da vrati najnoviji dump. Takođe, može da održi listu aktivnosti koje se mogu poništiti ili ponoviti (*undo* i *redo*). Još jedna od mogućnosti koje pruža oporavak je vraćanje stanja svih transakcija u trenutku poslednje kontrolne tačke.

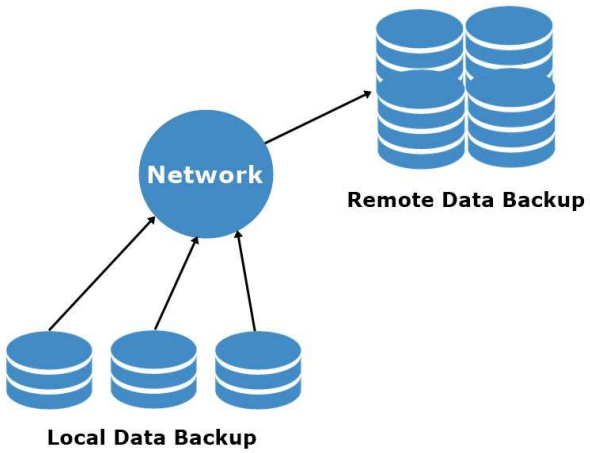
Katastrofalni kvar je kvar gde se sekundarni, stabilni uređaj za skladištenje ošteti. Sa uređajem za skladištenje gube se svi vredni podaci koji se na njemu čuvaju. Postoje dva načina oporavka od katastrofalnog kvara:

- **udaljene sigurnosne kopije** – čuvanje rezervne kopije na udaljenim lokacijama odakle se podaci mogu obnoviti u slučaju katastrofe;

- **magnetni diskovi** – čuvanje na magnetnim diskovima, odakle se sigurnosna kopija može preneti na sveže instaliranu bazu podataka.

Razrađene baze podataka su previše glomazne da bismo često skladištili njihove sigurnosne kopije. U takvim slučajevima koristimo tehnike gde možemo da obnovimo bazu podataka samo tako što ćemo pogledati dnevnik transakcija. Drugim rečima, umesto čuvanja cele baze podataka, možemo čuvati istoriju dnevnika sa izvršenim aktivnostima, što znatno smanjuje prostor u memoriji.

Udaljeno sigurnosno kopiranje pruža osećaj sigurnosti u slučaju da primarna lokacija na kojoj se nalazi baza bude uništena. Ove kopije mogu biti offline, u stvarnom vremenu ili na mreži. Bitno je napomenuti da se offline kopije ručno održavaju.



Slika 3.23. Prikaz udaljenog sigurnosnog kopiranja baze podataka

Mrežni sigurnosni sistemi se češće koriste u praksi i lakši su za upotrebu. **Sistem sigurnosnih kopija na mreži** je mehanizam u kojem se za svaki bit podataka u stvarnom vremenu pravi sigurnosna kopija istovremeno na dva mesta, u sistemu i na udaljenoj lokaciji. Čim primarna memorija baze podataka doživi prekid, sistem sigurnosnih kopija prebacuje korisnički sistem na udaljenu lokaciju za skladištenje. Ponekad je to izvršenje toliko brzo da korisnici ne mogu da primete kvar.

Oporavak baze podataka

SUBP je izuzetno složen sistem sa stotinama transakcija koje se izvršavaju svake sekunde. Trajnost i robusnost SUBP zavisi od njegove složene arhitekture i osnovnog hardvera i softvera sistema. Ako doživi neuspeh ili padne usred transakcija, očekuje se da će sistem slediti neku vrstu algoritma ili tehnika za vraćanje izgubljenih podataka.

Radi lakše detekcije problema, razlikujemo nekoliko kategorija neuspeha:

- neuspešne transakcije;
- pad sistema;
- kvarovi diska.

Neuspešne transakcije – transakcija doživljava neuspeh kada ne može da se izvrši ili dođe do određene tačke izvršavanja koja nije finalna. Razlozi za neuspeh transakcije mogu biti:

- **logičke greške** – greške nastale u kodu i druge interne greške;
- **sistemske greške** – kada sistem sam prekida transakciju jer SUBP ne može da je izvrši ili se mora zaustaviti zbog nekog drugog stanja sistema; uzrok ovoga su zastoj i nedostupnost resursa.

Pad sistema podrazumeva prekid sistema usled nekih spoljnih faktora koji utiču na sistem, kao što je npr. problem sa napajanjem.

Kvarovi diska – nedostupnost diska, oštećenje glave diska ili bilo koji drugi kvar koji dovodi do gubitka dela podataka ili svih podataka.

Strukturu memorije prema sigurnosti možemo podeliti u dve osnovne kategorije:

- **Nepostojana memorija** ne može da preživi pad sistema. Ovi uređaji za skladištenje se nalaze blizu procesora. Klasični primeri ove memorije su glavna memorija i *cache* memorija. Njihova prednost je brzina, a nedostatak skladištenje male količine podataka.
- **Postojana memorija** kreirana je da preživi pad sistema. Primer ovih uređaja su hard disk, magnetni diskovi, fleš memorija i sigurnosna RAM memorija. Njihova prednost je mogućnost skladištenja velike količine podataka, dok je glavni nedostatak spora pristupačnost podacima.

Kada se sistem sruši, moguće je izvršiti nekoliko transakcija i otvoriti razne datoteke za modifikovanje podataka. Transakcije su napravljene od različitih operacija, koje su atomične prirode. Prema ACID svojstvima SUBP, atomičnost transakcija u celini mora biti održavana – odnosno, ili će se izvršiti sve operacije, ili neće nijedna.

Kada se DBMS oporavi od pada, potrebno je sledeće:

- provera stanja svih transakcija koje su izvršene;
- osiguravanje atomičnosti u slučaju da je neka od transakcija prekinuta;
- provera da li transakcija može da se izvrši ili je treba prekinuti;
- nijedna transakcija ne sme da ostavi SUBP u nedoslednom stanju.

Postoje dve tehnike koje mogu pomoći SUBP u oporavku i u održavanju atomičnosti:

- održavanje dnevnika svake transakcije i njihovo zapisivanje u stabilnu memoriju pre nego što dođe do promena u bazi;
- održavanje nacrta, gde se promene vrše na nesigurnoj memoriji, a kasnije se ažurira stvarna baza podataka.

Oporavak na osnovu dnevnika transakcija vrši se na osnovu niza zapisa koji vode evidenciju akcija izvršenih transakcijama. Od izuzetne je važnosti vođenje zapisa i njihovo čuvanje na stabilnom mediju pre nego što same promene budu izvršene. Oporavak na osnovu dnevnika funkcioniše na sledeći način:

- fajl dnevnika se čuva na stabilnom mediju za skladištenje podataka;
- kada transakcija uđe u sistem i započne izvršenje, ona piše dnevnik o tome: <Tn, Start>;
- kada transakcija modifikuje stavku X, to zapisuje na sledeći način: <Tn, Ks, V1, V2>, gde V1 i V2 predstavljaju staru i novu vrednost za X;
- kada se transakcija izvrši, ispisuje se: <Tn, commit>.

Postoje dva pristupa modifikaciji baze podataka.

Odložena modifikacija podrazumeva da se svi zapisi upisuju u stabilnu memoriju i da se baza podataka ažurira kada se transakcija započne.

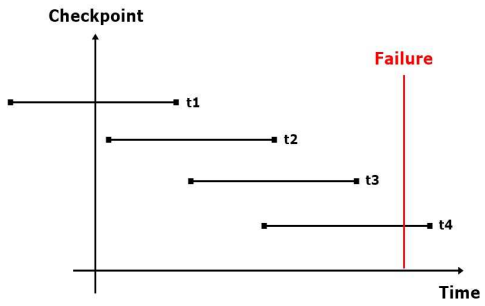
Trenutna modifikacija podrazumeva da svaki zapis prati stvarnu izmenu baze podataka. Odnosno, baza podataka se menja odmah nakon izvršenja svake operacije.

Oporavak paralelnih transakcija – kada se paralelno izvršava više transakcija, dnevnici se prepliću. U trenutku oporavka, sistemu za oporavak bilo bi teško da povuče sve zapise. Da bi se olakšala ova situacija, većina modernih SUBP koristi koncept kontrolnih tačaka.

Kontrolne tačke – čuvanje i održavanje zapisa u realnom vremenu i u stvarnom okruženju može popuniti sav memorijski prostor dostupan u sistemu. Kontrolne tačke predstavljaju mehanizam gde se svi prethodni zapisi uklanjaju iz sistema i trajno smeštaju na memorijski disk.

Kada se sistem sa kontrolnim tačkama naruši i oporavi, ponaša se na sledeći način:

- čita zapise unazad od kraja do poslednje kontrolne tačke;
- održava dve liste: listu za poništavanje i listu za ponavljanje (*undo* i *redo*);
- ako sistem za oporavak vidi zapis <Tn, Start> i <Tn, Commit> ili samo <Tn, Commit>, transakciju stavlja u *redo* listu;
- ako sistem za oporavak vidi zapis <Tn, Start>, ali nije pronađen dnevnik zapisivanja ili prekida, transakciju stavlja u *undo* listu.



Slika 3.24. Prikaz narušavanja sistema za vreme izvršavanja transakcije

Sada kada imamo jasnu sliku o tome čemu služe transakcije, kako se koriste i kakvu ulogu imaju u bazama podataka, prikazaćemo njihovo funkcionisanje na konkretnom primeru. Vratimo se našem primeru transakcije novca sa računa A na račun B. Ranije u lekciji smo

prikazali kako bi ovaj primer izgledao u objektno orijentisanom programiranju. Međutim, kako bi podaci u bazi bili ažurirani, odnosno, kako bi izmene nad njima bile trajne i u realnom vremenu, neophodno je da transakcije izvrše sve svoje operacije nad samom bazom u koju se podaci upisuju.

Za početak ćemo u program uvesti biblioteku koja nam je neophodna za rad nad bazom podataka korišćenjem komande `import sqlite3`. Zatim pravimo konekciju sa fajlom `accounts.db` i uvodimo `cursor`. U našem fajlu smo kreirali tabelu koja čuva podatke o vlasnicima računa i njihovom stanju na računu, u koju zatim upisujemo podatke o računima A i B. Ovi koraci su detaljno obrađeni u lekciji *Upravljanje bazom podataka* u kursu *Web Application Building*.

Identično kao i u našem primeru koji se bazira na objektno orijentisanom programiranju, zaposlenom A ćemo umanjiti stanje na računu za 100 USD, a zaposlenom B povećati stanje za isti iznos. Sada kada smo upisali sve komande koje su nam potrebne za izmene nad našim podacima u bazi, izvršićemo transakciju naredbom `conn.commit()`.

Pretpostavimo da se u trenutku odvijanja transakcije dogodi prekid nakon umanjenja stanja računa zaposlenog A. Ovako izvršena transakcija bi dovela do gubitka novca zaposlenog A, čiji novac nije dospeo do računa zaposlenog B. Iz ovih razloga uvodimo `try-catch` blok koji se stara o atomičnosti transakcije i koji dozvoljava da transakcija bude izvršena samo ukoliko su sve njene operacije izvršene. U slučaju da transakcija ne može u potpunosti da se izvrši, poništavamo je komandom `conn.rollback()`.

Prvim pokretanjem koda u našem primeru kao rezultat ćemo dobiti uspešno izvršenje transakcije. Međutim, drugim pokretanjem transakcija neće uspeti da se izvrši. Razlog tome su definisani primarni ključevi koji moraju biti jedinstveni. Drugim rečima, sa drugim pokretanjem naš kod ponovo pokušava da umetne zaposlene čija imena A i B predstavljaju primarne ključeve zapisa, te iz tog razloga transakcija neće proći.

Radno okruženje

```
import sqlite3

conn = sqlite3.connect('accounts.db')
cursor = conn.cursor()

try:
    cursor.execute("CREATE TABLE IF NOT EXISTS accounts ("
                   "name TEXT NOT NULL PRIMARY KEY,balance FLOAT NOT
NULL)")
    account = [('A',100.0), ('B', 100.0)]
    cursor.executemany("INSERT INTO accounts VALUES (?, ?)", account)
    cursor.execute("UPDATE accounts SET balance = balance - 100.0 "
     "WHERE name = 'A'")
    cursor.execute("UPDATE accounts SET balance = balance + 100.0 "
     "WHERE name = 'B'")
    conn.commit()
    print('Transaction committed!')

except conn.Error:
    conn.rollback()
    print('Transaction rolledback!')
```

Izmenite kod tako da postoje 3 računa i svaki od njih na stanju ima isti iznos od 5000 dinara. Prebacite iznos od 2000 sa računa A na račun B, sa računa B na račun C 4000 a sa računa C na račun A 6000.

Radno okruženje

Rezime

- Memoriju po kapacitetu i brzini pristupa možemo podeliti u tri kategorije:
    - primarna – najbrža memorija, direktno dostupna procesoru;
    - sekundarna – pravi rezervne kopije podataka za buduće korišćenje;
    - tercijarna – skladišti ogromnu količinu podataka uz malu brzinu pristupa.

- RAID je tehnologija za povezivanje više sekundarnih uređaja za skladištenje podataka u jedinstveni medijum.
- Organizacija fajlova na disku može biti: heap, sekvencijalna, hash i klasterovana.
- Osnovne operacije nad fajlovima su `open()`, `locate()`, `read()`, `write()` i `close()`.
- Indeksiranje je tehnika struktura podataka za efikasno pronalaženje zapisa na osnovu nekih atributa nad kojima je vršeno indeksiranje.
- Višeslojni indeksi koriste se kada je veličina indeksa veća od one koja može da se skladišti na memoriji.
- Hashovanje je efikasna tehnika za računanje direktne lokacije traženog podatka na disku bez korišćenja tehnike indeksiranja. Može biti statičko i dinamičko.
- Transakcija podrazumeva grupu zadataka gde pojedinačni zadatak predstavlja minimalnu procesnu jedinicu, koja se ne može dalje deliti.
- ACID osobine transakcije su:
    - atomičnost (atomicity) – izvršavaju se ili sve operacije ili nijedna;
    - konzistentnost (consistency) – doslednost baze podataka pre i nakon izvršenja transakcije;
    - izolovanost (isolation) – efekti transakcija koje se obavljaju istovremeno su izolovani;
    - trajnost (durability) – efekti transakcija su trajni na bazi podataka.

- Serijabilnost podrazumeva izvršavanje više transakcija u okruženju sa više programa, što dovodi do konflikta transakcija.
- Moguća stanja transakcija su: aktivna, delimično izvršena, neuspešna, prekinuta i izvršena.
- Dva načina izvršavanja paralelnih transakcija su:
    - protokoli zaključavanja – postavljanje lokota na podatak pre izvršavanja transakcije;
    - timestamp protokoli – vremensko označavanje transakcija po redosledu izvršavanja.

- Sigurnosna kopija podataka koristi se u slučaju narušavanja primarne memorije koja nije dovoljno stabilna. Neke od tehnika su: uvođenje kontrolnih tačaka, automatsko čuvanje i `<dump>` markeri
- Dve osnovne operacije transakcija su:

    - `commit()` – trajno čuva sve promene nad podacima u bazi;
    - `rollback()` – poništava sve započete operacije trenutne transakcije i vraća bazu podataka u stanje u kojem je bila pre pokretanja te transakcije.